

Efficient Lyndon factorization of grammar compressed text

Tomohiro I^{1,2}, Yuto Nakashima¹, Shunsuke Inenaga¹, Hideo Bannai¹, and
Masayuki Takeda¹

¹ Department of Informatics, Kyushu University, Japan
{tomohiro.i, yuto.nakashima, inenaga, bannai, takeda}@inf.kyushu-u.ac.jp

² Japan Society for the Promotion of Science (JSPS)

Abstract. We present an algorithm for computing the Lyndon factorization of a string that is given in grammar compressed form, namely, a Straight Line Program (SLP). The algorithm runs in $O(n^4 + mn^3h)$ time and $O(n^2)$ space, where m is the size of the Lyndon factorization, n is the size of the SLP, and h is the height of the derivation tree of the SLP. Since the length of the decompressed string can be exponentially large w.r.t. n, m and h , our result is the first polynomial time solution when the string is given as SLP.

1 Introduction

Compressed string processing (CSP) is a task of processing compressed string data without explicit decompression. As any method that first decompresses the data requires time and space dependent on the decompressed size of the data, CSP without explicit decompression has been gaining importance due to the ever increasing amount of data produced and stored. A number of efficient CSP algorithms have been proposed, e.g., see [16,25,15,12,11,13]. In this paper, we present new CSP algorithms that compute the *Lyndon factorization* of strings.

A string ℓ is said to be a *Lyndon word* if ℓ is lexicographically smallest among its circular permutations of characters of ℓ . For example, **aab** is a Lyndon word, but its circular permutations **aba** and **baa** are not. Lyndon words have various and important applications in, e.g., musicology [4], bioinformatics [8], approximation algorithm [22], string matching [6,2,23], word combinatorics [10,24], and free Lie algebras [20].

The *Lyndon factorization* (a.k.a. *standard factorization*) of a string w , denoted $LF(w)$, is a unique sequence of Lyndon words such that the concatenation of the Lyndon words gives w and the Lyndon words in the sequence are lexicographically non-increasing [5]. Lyndon factorizations are used in a bijective variant of Burrows-Wheeler transform [17,14] and a digital geometry algorithm [3]. Duval [9] proposed an elegant on-line algorithm to compute $LF(w)$ of a given string w of length N in $O(N)$ time. Efficient parallel algorithms to compute the Lyndon factorization are also known [1,7].

We present a new CSP algorithm which computes the Lyndon factorization $LF(w)$ of a string w , when w is given in a *grammar-compressed form*. Let m

be the number of factors in $LF(w)$. Our first algorithm computes $LF(w)$ in $O(n^4 + mn^3h)$ time and $O(n^2)$ space, where n is the size of a given *straight-line program* (SLP), which is a context-free grammar in Chomsky normal form that derives only w , and h is the height of the derivation tree of the SLP. Since the decompressed string length $|w| = N$ can be exponentially large w.r.t. n, m and h , our $O(n^4 + mn^3h)$ solution can be efficient for highly compressive strings.

2 Preliminaries

2.1 Strings and model of computation

Let Σ be a finite *alphabet*. An element of Σ^* is called a *string*. The length of a string w is denoted by $|w|$. The empty string ε is a string of length 0, namely, $|\varepsilon| = 0$. Let Σ^+ be the set of non-empty strings, i.e., $\Sigma^+ = \Sigma^* - \{\varepsilon\}$. For a string $w = xyz$, x , y and z are called a *prefix*, *substring*, and *suffix* of w , respectively. A prefix x of w is called a *proper prefix* of w if $x \neq w$, i.e., x is shorter than w . The set of suffixes of w is denoted by $Suffix(w)$. The i -th character of a string w is denoted by $w[i]$, where $1 \leq i \leq |w|$. For a string w and two integers $1 \leq i \leq j \leq |w|$, let $w[i..j]$ denote the substring of w that begins at position i and ends at position j . For convenience, let $w[i..j] = \varepsilon$ when $i > j$. For any string w let $w^1 = w$, and for any integer $k > 2$ let $w^k = ww^{k-1}$, i.e., w^k is a k -time repetition of w .

A positive integer p is said to be a *period* of a string w if $w[i] = w[i + p]$ for all $1 \leq i \leq |w| - p$. Let w be any string and q be its smallest period. If p is a period of a string w such that $p < |w|$, then the positive integer $|w| - p$ is said to be a *border* of w . If w has no borders, then w is said to be *border-free*.

If character $a \in \Sigma$ is lexicographically smaller than another character $b \in \Sigma$, then we write $a \prec b$. For any non-empty strings $x, y \in \Sigma^+$, let $lcp(x, y)$ be the length of the longest common prefix of x and y . We denote $x \prec y$, if either of the following conditions holds: $x[lcp(x, y) + 1] \prec y[lcp(x, y) + 1]$, or x is a proper prefix of y . For a set $S \subseteq \Sigma^+$ of non-empty strings, let $\min_{\prec} S$ denote the lexicographically smallest string in S .

Our model of computation is the word RAM: We shall assume that the computer word size is at least $\lceil \log_2 |w| \rceil$, and hence, standard operations on values representing lengths and positions of string w can be manipulated in constant time. Space complexities will be determined by the number of computer words (not bits).

2.2 Lyndon words and Lyndon factorization of strings

Two strings x and y are said to be *conjugate*, if there exist strings u and v such that $x = uv$ and $y = vu$. A string w is said to be a *Lyndon word*, if w is lexicographically strictly smaller than all of its conjugates of w . Namely, w is a Lyndon word, if for any factorization $w = uv$, it holds that $uv \prec vu$. It is known that any Lyndon word is border-free.

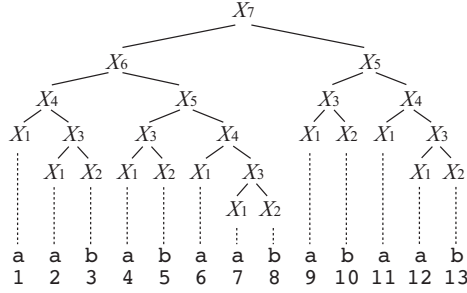


Fig. 1. The derivation tree of SLP $\mathcal{S} = \{X_1 \rightarrow a, X_2 \rightarrow b, X_3 \rightarrow X_1X_2, X_4 \rightarrow X_1X_3, X_5 \rightarrow X_3X_4, X_6 \rightarrow X_4X_5, X_7 \rightarrow X_6X_5\}$, representing string $S = \text{val}(X_7) = \text{aababaababab}$.

Definition 1 ([5]). The Lyndon factorization of a string w , denoted $LF(w)$, is the factorization $\ell_1^{p_1} \dots \ell_m^{p_m}$ of w , such that each $\ell_i \in \Sigma^+$ is a Lyndon word, $p_i \geq 1$, and $\ell_i \succ \ell_{i+1}$ for all $1 \leq i < m$.

It is known that the Lyndon factorization is unique for each string w , and it was shown by Duval [9] that the Lyndon factorization can be computed in $O(N)$ time, where $N = |w|$.

$LF(w)$ can be represented by the sequence $(|\ell_1|, p_1), \dots, (|\ell_m|, p_m)$ of integer pairs, where each pair $(|\ell_i|, p_i)$ represents the i -th Lyndon factor $\ell_i^{p_i}$ of w . Note that this representation requires $O(m)$ space.

2.3 Straight line programs

A *straight line program (SLP)* is a set of productions $\mathcal{S} = \{X_1 \rightarrow \text{expr}_1, X_2 \rightarrow \text{expr}_2, \dots, X_n \rightarrow \text{expr}_n\}$, where each X_i is a variable and each expr_i is an expression, where $\text{expr}_i = a$ ($a \in \Sigma$), or $\text{expr}_i = X_{\ell(i)}X_{r(i)}$ ($i > \ell(i), r(i)$). It is essentially a context free grammar in Chomsky normal form, that derives a single string. Let $\text{val}(X_i)$ represent the string derived from variable X_i . To ease notation, we sometimes associate $\text{val}(X_i)$ with X_i and denote $|\text{val}(X_i)|$ as $|X_i|$, and $\text{val}(X_i)[u..v]$ as $X_i[u..v]$ for $1 \leq u \leq v \leq |X_i|$. An SLP \mathcal{S} represents the string $w = \text{val}(X_n)$. The *size* of the program \mathcal{S} is the number n of productions in \mathcal{S} . Let N be the length of the string represented by SLP \mathcal{S} , i.e., $N = |w|$. Then N can be as large as 2^{n-1} .

The derivation tree of SLP \mathcal{S} is a labeled ordered binary tree where each internal node is labeled with a non-terminal variable in $\{X_1, \dots, X_n\}$, and each leaf is labeled with a terminal character in Σ . The root node has label X_n . An example of the derivation tree of an SLP is shown in Fig. 1.

3 Computing Lyndon factorization from SLP

In this section, we show how, given an SLP \mathcal{S} of n productions representing string w , we can compute $LF(w)$ of size m in $O(n^4 + mn^3h)$ time. We will make use of the following known results:

Lemma 1 ([9]). *For any string w , let $LF(w) = \ell_1^{p_1}, \dots, \ell_m^{p_m}$. Then, $\ell_m = \min_{\prec} \text{Suffix}(w)$, i.e., ℓ_m is the lexicographically smallest suffix of w .*

Lemma 2 ([18]). *Given an SLP \mathcal{S} of size n representing a string w of length N , and two integers $1 \leq i \leq j \leq N$, we can compute in $O(n)$ time another SLP of size $O(n)$ representing the substring $w[i..j]$.*

Lemma 3 ([18]). *Given an SLP \mathcal{S} of size n representing a string w of length N , we can compute the shortest period of w in $O(n^3 \log N)$ time and $O(n^2)$ space.*

For any non-empty string $w \in \Sigma^+$, let $LF\text{Cand}(w) = \{x \mid x \in \text{Suffix}(w), \exists y \in \Sigma^+ \text{ s.t. } xy = \min_{\prec} \text{Suffix}(wy)\}$. Intuitively, $LF\text{Cand}(w)$ is the set of suffixes of w which are a prefix of the lexicographically smallest suffix of string wy , for some non-empty string $y \in \Sigma^+$.

The following lemma may be almost trivial, but will play a central role in our algorithm.

Lemma 4. *For any two strings $u, v \in LF\text{Cand}(w)$ with $|u| < |v|$, u is a prefix of v .*

Proof. If $v[1..|u|] \prec u$, then for any non-empty string y , $vy \prec uy$. However, this contradicts that $u \in LF\text{Cand}(w)$. If $v[1..|u|] \succ u$, then for any non-empty string y , $vy \succ uy$. However, this contradicts that $v \in LF\text{Cand}(w)$. Hence we have $v[1..|u|] = u$. \square

Lemma 5. *For any string w , let $\ell = \min_{\prec} \text{Suffix}(w)$. Then, the shortest string of $LF\text{Cand}(w)$ is ℓ^p , where $p \geq 1$ is the maximum integer such that ℓ^p is a suffix of w .*

Proof. For any string $x \in LF\text{Cand}(w)$, and any non-empty string y , $xy = \min_{\prec} \text{Suffix}(wy)$ holds only if $y \succ \ell$.

Firstly, we compare ℓ^p with the suffixes s of w shorter than ℓ^p , and show that $\ell^p y \prec sy$ holds for any $y \succ \ell$. Such suffixes s are divided into two groups: (1) If s is of form ℓ^k for any integer $1 \leq k < p$, then $\ell^p y \prec \ell^k y = sy \prec y$ holds for any $y \succ \ell$; (2) If s is not of form ℓ^k , then since ℓ is border-free, ℓ is not a prefix of s , and s is not a prefix of ℓ , either. Thus $\ell^p \prec s$ holds, implying that $\ell^p y \prec sy$ for any $y \succ \ell$.

Secondly, we compare ℓ^p with the suffixes t of w longer than ℓ^p , and show that $\ell^p y \prec ty$ holds for some $y \succ \ell$. By Lemma 4, $t = \ell^q u$ holds, where $q \geq p$ is the maximum integer such that ℓ^q is a prefix of t , and $u \in \Sigma^+$. By definition, $\ell \prec u$ and ℓ is not a prefix of u . Choosing $y = \ell^{q-p} u'$ with $u' \prec u$, we have $\ell^p y = \ell^q u' \prec \ell^q u = t \prec ty$. Hence, $\ell^p \in LF\text{Cand}(w)$ and no shorter strings exist in $LF\text{Cand}(w)$. \square

By Lemma 1 and Lemma 5, computing the last Lyndon factor $\ell_m^{p_m}$ of $w = \text{val}(X_n)$ reduces to computing $\text{LFCand}(X_n)$ for the last variable X_n . In what follows, we propose a dynamic programming algorithm to compute $\text{LFCand}(X_i)$ for each variable. Firstly we show the number of strings in $\text{LFCand}(X_i)$ is $O(\log N)$, where $N = |\text{val}(X_n)| = |w|$.

Lemma 6. *For any string w , let s_j be the j th shortest string of $\text{LFCand}(w)$. Then, $|s_{j+1}| > 2|s_j|$ for any $1 \leq j < |\text{LFCand}(w)|$.*

Proof. Let $\ell = \min_{\prec} \text{Suffix}(w)$, and y any string such that $y \succ \ell$. It follows from Lemma 4 that ℓ is a prefix of any string $s_j \in \text{LFCand}(w)$, and hence $s_j \prec y$ holds.

Assume on the contrary that $|s_{j+1}| \leq 2|s_j|$. If $|s_{j+1}| = 2|s_j|$, i.e., $s_{j+1} = s_j s_j$, then $s_{j+1}y = s_j s_j y \prec s_j y$ holds, but this contradicts that $s_j \in \text{LFCand}(w)$. Hence $s_{j+1} \neq s_j s_j$. If $|s_{j+1}| < 2|s_j|$, by Lemma 4, s_j is a prefix of s_{j+1} , and therefore s_j has a period q such that $s_{j+1} = u^k v$ and $s_j = u^{k-1} v$, where $u = s_j[1..q]$, $k \geq 1$ is an integer, and v is a proper prefix of u . There are two cases to consider: (1) If $uvy \prec vy$, then $u^k v y \prec u^{k-1} v y = s_j y$. (2) If $vy \prec uv y$, then $vy \prec uv y \prec u^2 v y \prec \dots \prec u^{k-1} v y = s_j y$. It means that $\min_{\prec} \{u^k v y, v y\} \prec s_j y$ for any $y \succ \ell$, however, this contradicts that $s_j \in \text{LFCand}(w)$. Hence $|s_{j+1}| > 2|s_j|$ holds. \square

Since s_j is a suffix of s_{j+1} , it follows from Lemma 4 and Lemma 6 that $s_{j+1} = s_j t s_j$ with some non-empty string $t \in \Sigma^+$. This also implies that the number of strings in $\text{LFCand}(w)$ is $O(\log N)$, where N is the length of w . By identifying each suffix of $\text{LFCand}(X_i)$ with its length, and using Lemma 6, $\text{LFCand}(X_i)$ for all variables can be stored in a total of $O(n \log N)$ space.

For any two variables X_i, X_j of an SLP \mathcal{S} and a positive integer k satisfying $|X_i| \geq k + |X_j| - 1$, consider the FM function such that $FM(X_i, X_j, k) = \text{lcp}(\text{val}(X_i)[k..|X_i|], \text{val}(X_j))$, i.e., it returns the length of the lcp of the suffix of $\text{val}(X_i)$ starting at position k and X_j .

Lemma 7 ([21,19]). *We can preprocess a given SLP \mathcal{S} of size n in $O(n^3)$ time and $O(n^2)$ space so that $FM(X_i, X_j, k)$ can be answered in $O(n^2)$ time.*

For each variable X_i we store the length $|X_i|$ of the string derived by X_i . It requires a total of $O(n)$ space for all $1 \leq i \leq n$, and can be computed in a total of $O(n)$ time by a simple dynamic programming algorithm. Given a position j of the uncompressed string w of length N , i.e., $1 \leq j \leq N$, we can retrieve the j th character $w[j]$ in $O(n)$ time by a simple binary search on the derivation tree of X_n using the lengths stored in the variables. Hence, we can lexicographically compare $\text{val}(X_i)[k..|X_i|]$ and $\text{val}(X_j)$ in $O(n^2)$ time, after $O(n^3)$ -time preprocessing.

The following lemma shows a dynamic programming approach to compute $\text{LFCand}(X_i)$ for each variable X_i . We will mean by a sorted list of $\text{LFCand}(X_i)$ the list of the elements of $\text{LFCand}(X_i)$ sorted in increasing order of length.

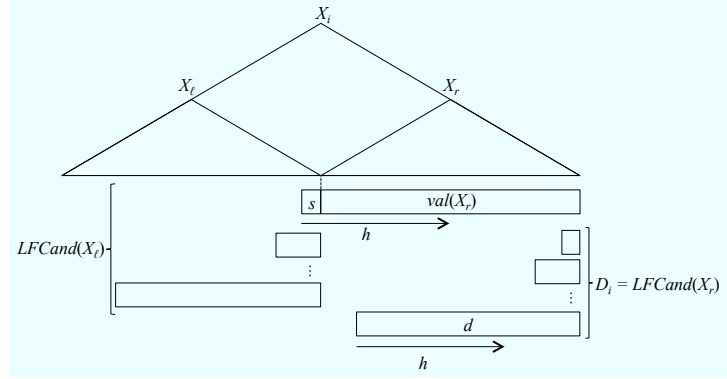


Fig. 2. Lemma 8: Initially $D_i = LFCand(X_r)$ and $h = s \cdot val(X_\ell)$ with s being the shortest string of $LFCand(X_\ell)$.

Lemma 8. Let $X_i = X_\ell X_r$ be any production of a given SLP \mathcal{S} of size n . Provided that sorted lists for $LFCand(X_\ell)$ and $LFCand(X_r)$ are already computed, a sorted list for $LFCand(X_i)$ can be computed in $O(n^3)$ time and $O(n^2)$ space.

Proof. Let D_i be a sorted list of the suffixes of X_i that are candidates of elements of $LFCand(X_i)$. We initially set $D_i \leftarrow LFCand(X_r)$.

We process the elements of $LFCand(X_\ell)$ in increasing order of length. Let s be any string in $LFCand(X_\ell)$, and d the longest string in D_i . Since any string of $LFCand(X_r)$ is a prefix of d by Lemma 4, in order to compute $LFCand(X_i)$ it suffices to lexicographically compare $s \cdot val(X_r)$ and d . Let $h = lcp(s \cdot val(X_r), d)$. See also Fig. 2.

- If $(s \cdot val(X_r))[h+1] < d[h+1]$, then $s \cdot val(X_r) < d$. Since any string in D_i is a prefix of d by Lemma 4, we observe that any element in D_i that is longer than h cannot be an element of $LFCand(X_i)$. Hence we delete any element of D_i that is longer than h from D_i , then add $s \cdot val(X_r)$ to D_i , and update $d \leftarrow s \cdot val(X_r)$. See also Fig. 3.
- If $(s \cdot val(X_r))[h+1] > d[h+1]$, then $s \cdot val(X_r) > d$. Since $s \cdot val(X_r)$ cannot be an element of $LFCand(X_i)$, in this case neither D_i nor d is updated. See also Fig. 4.
- If $h = |d|$, i.e., d is a prefix of $s \cdot val(X_r)$, then there are two sub-cases:
 - If $|s \cdot val(X_r)| \leq 2|d|$, d has a period q such that $s \cdot val(X_r) = u^k v$ and $d = u^{k-1} v$, where $u = d[1..q]$, $k \geq 1$ is an integer, and v is a proper prefix of u . By similar arguments to Lemma 6, we observe that d cannot be a member of $LFCand(X_i)$ while $s \cdot val(X_r)$ may be a member of $LFCand(X_i)$. Thus we add $s \cdot val(X_r)$ to D_i , delete d from D_i , and update $d \leftarrow s \cdot val(X_r)$. See also Fig. 5.
 - If $|s \cdot val(X_r)| > 2|d|$, then both d and $s \cdot val(X_r)$ may be a member of $LFCand(X_i)$. Thus we add $s \cdot val(X_r)$ to D_i , and update $d \leftarrow s \cdot val(X_r)$. See also Fig. 6.

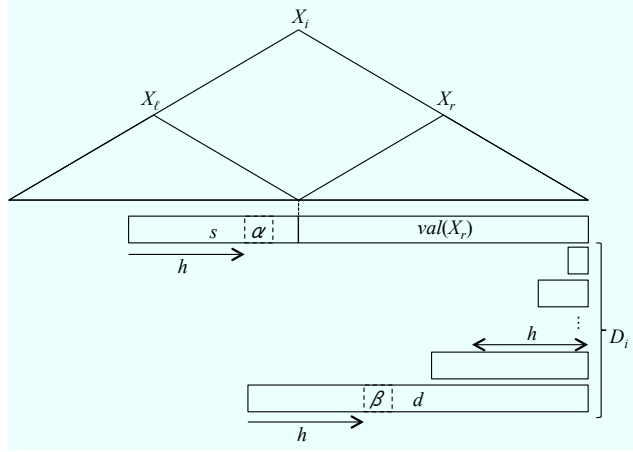


Fig. 3. Lemma 8: Case where $(s \cdot \text{val}(X_r))[h+1] = \alpha < d[h+1] = \beta$. d and any string in D_i that is longer than h are deleted from D_i . Then $s \cdot \text{val}(X_r)$ becomes the longest candidate in D_i .

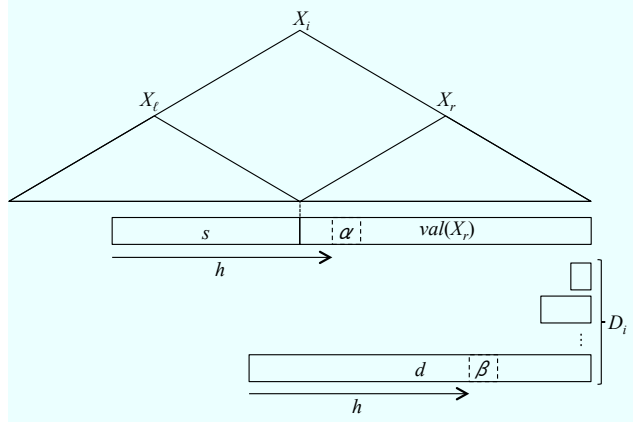


Fig. 4. Lemma 8: Case where $(s \cdot \text{val}(X_r))[h+1] = \alpha > d[h+1] = \beta$. There are no updates on D_i .

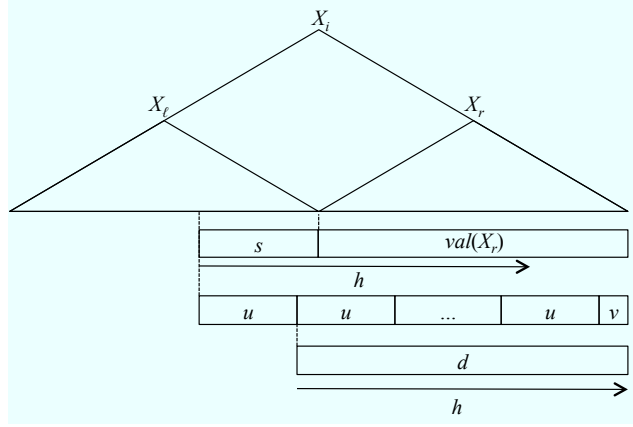


Fig. 5. Lemma 8: Case where $h = |d|$ and $|s \cdot val(X_r)| \leq 2|d|$. Since $s \cdot val(X_r) = u^k v$ and $d = u^{k-1} v$, d is deleted from D_i and $s \cdot val(X_r)$ is added to D_i .

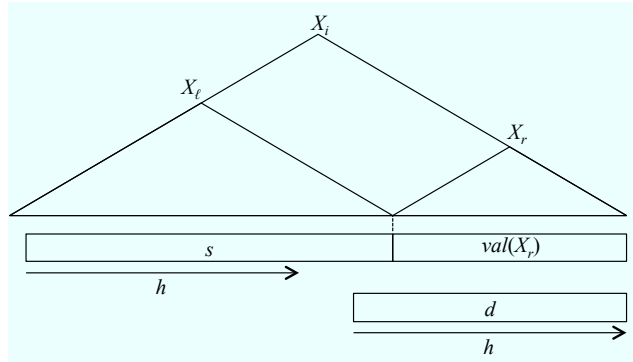


Fig. 6. Lemma 8: Case where $h = |d|$ and $|s \cdot val(X_r)| > 2|d|$. We add $s \cdot val(X_r)$ to D_i , and $s \cdot val(X_r)$ becomes the longest member of D_i .

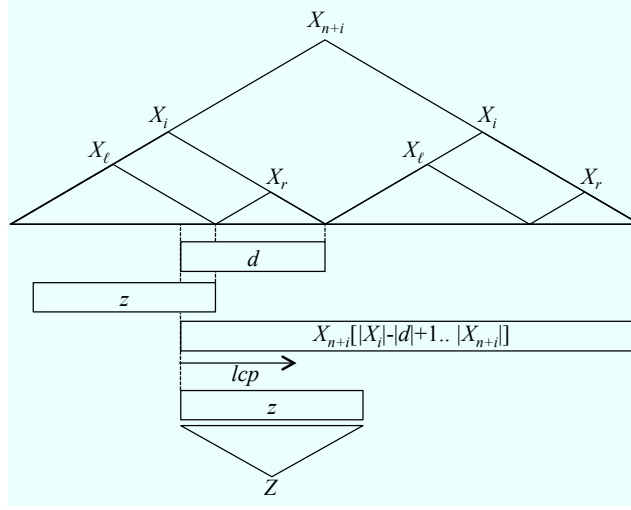


Fig. 7. Lemma 8: $lcp(z, d) = \min\{lcp(Z, X_{n+i}[|X_i| - |d| + 1 .. |X_{n+i}|]), |d|\}$.

We represent the strings in $LFCand(X_\ell)$, $LFCand(X_r)$, $LFCand(X_i)$, and D_i by their lengths. Given sorted lists of $LFCand(X_\ell)$ and $LFCand(X_r)$, the above algorithm computes a sorted list for D_i , and it follows from Lemma 6 that the number of elements in D_i is always $O(\log N)$. Thus all the above operations on D_i can be conducted in $O(\log N)$ time in each step.

We now show how to efficiently compute $h = lcp(s \cdot val(X_r), d)$, for any $s \in LFCand(X_\ell)$. Let z be the longest string in $LFCand(X_\ell)$, and consider to process any string $s \in LFCand(X_\ell)$. Since s is a prefix of z by Lemma 4, we can compute $lcp(s \cdot val(X_r), d)$ as follows:

$$lcp(s \cdot val(X_r), d) = \begin{cases} lcp(z, d) & \text{if } lcp(z, d) < |s|, \\ |s| + lcp(X_r, d[|s| + 1 .. |d|]) & \text{if } lcp(z, d) \geq |s|. \end{cases}$$

To compute the above lcp values using the FM function, for each variable X_i of \mathcal{S} we create a new production $X_{n+i} = X_i X_i$, and hence the number of variables increases to $2n$. In addition, we construct a new SLP of size $O(n)$ that derives z in $O(n)$ time using Lemma 2. Let Z be the variable such that $val(Z) = z$. It holds that

$$lcp(z, d) = \min\{lcp(Z, X_{n+i}[|X_i| - |d| + 1 .. |X_{n+i}|]), |d|\} \text{ and } lcp(X_r, d[|s| + 1 .. |d|]) = \min\{lcp(X_r, X_{n+r}[|X_r| - |d| + |s| + 1 .. |X_{n+r}|]), |d| - |s|\}.$$

See also Fig. 7 and Fig. 8.

By using Lemma 7, we preprocess, in $O(n^3)$ time and $O(n^2)$ space, the SLP consisting of these variables so that the query $FM(X_i, X_j, k)$ for answering $lcp(X_i[k .. |X_i|], X_j)$ is supported in $O(n^2)$ time. Therefore $lcp(s \cdot val(X_r), d)$ can

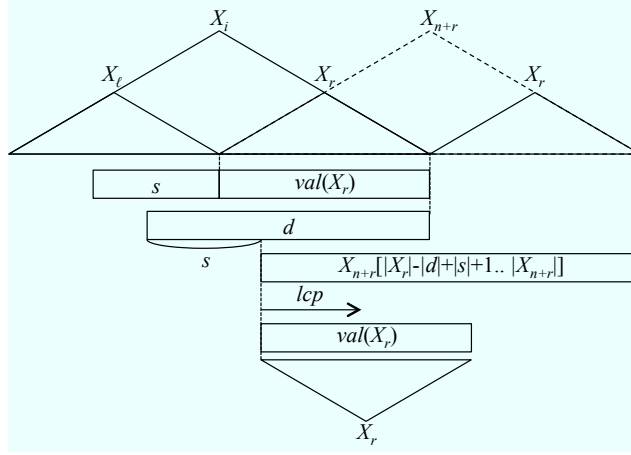


Fig. 8. Lemma 8: $lcp(X_r, d[|s| + 1..|d|]) = \min\{lcp(X_r, X_{n+r}[|X_r| - |d| + |s| + 1..|X_{n+r}|]), |d| - |s|\}$.

be computed in $O(n^2)$ time for each $s \in LFCand(X_\ell)$. Since there exist $O(\log N)$ elements in $LFCand(X_\ell)$, we can compute $LFCand(X_i)$ in $O(n^3 + n^2 \log N) = O(n^3)$ time. The total space complexity is $O(n^2)$. \square

Since there are n productions in a given SLP, using Lemma 8 we can compute $LFCand(X_n)$ for the last variable X_n in a total of $O(n^4)$ time. The main result of this paper follows.

Theorem 1. *Given an SLP \mathcal{S} of size n representing a string w , we can compute $LF(w)$ in $O(n^4 + mn^3h)$ time and $O(n^2)$ space, where m is the number of factors in $LF(w)$ and h is the height of the derivation tree of \mathcal{S} .*

Proof. Let $LF(w) = \ell_1^{p_1} \dots \ell_m^{p_m}$. First, using Lemma 8 we compute $LFCand$ for all variables in \mathcal{S} in $O(n^4)$ time. Next we will compute the Lyndon factors from right to left. Suppose that we have already computed $\ell_{j+1}^{p_{j+1}} \dots \ell_m^{p_m}$, and we are computing the j th Lyndon factor $\ell_j^{p_j}$. Using Lemma 2, we construct in $O(n)$ time a new SLP of size $O(n)$ describing $w[1..|w| - \sum_{k=j+1}^m p_k |\ell_k|]$, which is the prefix of w obtained by removing the suffix $\ell_{j+1}^{p_{j+1}} \dots \ell_m^{p_m}$ from w . Here we note that the new SLP actually has $O(h)$ new variables since $w[1..|w| - \sum_{k=j+1}^m p_k |\ell_k|]$ can be represented by a sequence of $O(h)$ variables in \mathcal{S} . Let Y be the last variable of the new SLP. Since $LFCand$ for all variables in \mathcal{S} have already been computed, it is enough to compute $LFCand$ for $O(h)$ new variables. Hence using Lemma 8, we compute a sorted list of $LFCand(Y) = LFCand(w[1..|w| - \sum_{k=j+1}^m p_k |\ell_k|])$ in a total of $O(n^3h)$ time. It follows from Lemma 5 that the shortest element of $LFCand(Y)$ is $\ell_j^{p_j}$, the j th Lyndon factor of w . Note that each string in $LFCand(Y)$ is represented by its length, and so far we only know the total length $p_j |\ell_j|$ of the j th Lyndon factor. Since ℓ_j is border free, $|\ell_j|$ is the shortest period

of $\ell_j^{p_j}$. We construct a new SLP of size $O(n)$ describing $\ell_j^{p_j}$, and compute $|\ell_j|$ in $O(n^3 \log N)$ time using Lemma 3. We repeat the above procedure m times, and hence $LF(w)$ can be computed in a total of $O(n^4 + m(n^3 h + n^3 \log N)) = O(n^4 + mn^3 h)$ time. To compute each Lyndon factor of $LF(w)$, we need $O(n^2)$ space for Lemma 3 and Lemma 8. Since $LF\text{Cand}(X_i)$ for each variable X_i requires $O(\log N)$ space, the total space complexity is $O(n^2 + n \log N) = O(n^2)$. \square

4 Conclusions and open problem

Lyndon words and Lyndon factorization are important concepts of combinatorics on words, with various applications. Given a string in terms of an SLP of size n , we showed how to compute the Lyndon factorization of the string in $O(n^4 + mn^3 h)$ time using $O(n^2)$ space, where m is the size of the Lyndon factorization and h is the height of the SLP. Since the decompressed string length N can be exponential w.r.t. n, m and h , our algorithm can be useful for highly compressive strings.

An interesting open problem is to compute the Lyndon factorization from a given LZ78 encoding [26]. Each LZ78 factor is a concatenation of the longest previous factor and a single character. Hence, it can be seen as a special class of SLPs, and this property would lead us to a much simpler and/or more efficient solution to the problem. Noting the number s of the LZ78 factors is $\Omega(\sqrt{N})$, a question is whether we can solve this problem in $o(s^2) + O(m)$ time.

References

1. Apostolico, A., Crochemore, M.: Fast parallel Lyndon factorization with applications. *Mathematical Systems Theory* 28(2), 89–108 (1995)
2. Breslauer, D., Grossi, R., Mignosi, F.: Simple real-time constant-space string matching. In: *Proc. CPM 2011*. pp. 173–183 (2011)
3. Brlek, S., Lachaud, J.O., Provençal, X., Reutenauer, C.: Lyndon + Christoffel = digitally convex. *Pattern Recognition* 42(10), 2239–2246 (2009)
4. Chemillier, M.: Periodic musical sequences and Lyndon words. *Soft Comput.* 8(9), 611–616 (2004)
5. Chen, K.T., Fox, R.H., Lyndon, R.C.: Free differential calculus. iv. the quotient groups of the lower central series. *Annals of Mathematics* 68(1), 81–95 (1958)
6. Crochemore, M., Perrin, D.: Two-way string matching. *J. ACM* 38(3), 651–675 (1991)
7. Daykin, J.W., Iliopoulos, C.S., Smyth, W.F.: Parallel RAM algorithms for factorizing words. *Theor. Comput. Sci.* 127(1), 53–67 (1994)
8. Delgrange, O., Rivals, E.: STAR: an algorithm to search for tandem approximate repeats. *Bioinformatics* 20(16), 2812–2820 (2004)
9. Duval, J.P.: Factorizing words over an ordered alphabet. *J. Algorithms* 4(4), 363–381 (1983)
10. Fredricksen, H., Maiorana, J.: Necklaces of beads in k colors and k -ary de Bruijn sequences. *Discrete Mathematics* 23(3), 207–210 (1978)
11. Gawrychowski, P.: Optimal pattern matching in LZW compressed strings. In: *Proc. SODA 2011*. pp. 362–372 (2011)

12. Gawrychowski, P.: Pattern matching in Lempel-Ziv compressed strings: Fast, simple, and deterministic. In: Proc. ESA 2011. pp. 421–432 (2011)
13. Gawrychowski, P.: Faster algorithm for computing the edit distance between SLP-compressed strings. In: Proc. SPIRE 2012. pp. 229–236 (2012)
14. Gil, J.Y., Scott, D.A.: A bijective string sorting transform. CoRR abs/1201.3077 (2012)
15. Goto, K., Bannai, H., Inenaga, S., Takeda, M.: Fast q -gram mining on SLP compressed strings. *Journal of Discrete Algorithms* 18, 89–99 (2013)
16. Hermelin, D., Landau, G.M., Landau, S., Weimann, O.: A unified algorithm for accelerating edit-distance computation via text-compression. In: Proc. STACS 2009. pp. 529–540 (2009)
17. Kufleitner, M.: On bijective variants of the Burrows-Wheeler transform. In: Proc. PSC 2009. pp. 65–79 (2009)
18. Lifshits, Y.: Solving classical string problems on compressed texts. In: *Combinatorial and Algorithmic Foundations of Pattern and Association Discovery*. No. 06201 in Dagstuhl Seminar Proceedings (2006)
19. Lifshits, Y.: Processing compressed texts: A tractability border. In: Proc. CPM 2007. LNCS, vol. 4580, pp. 228–240 (2007)
20. Lyndon, R.C.: On Burnside’s problem. *Transactions of the American Mathematical Society* 77, 202–215 (1954)
21. Miyazaki, M., Shinohara, A., Takeda, M.: An improved pattern matching algorithm for strings in terms of straight-line programs. In: Proc. CPM1997. LNCS, vol. 1264, pp. 1–11 (1997)
22. Mucha, M.: Lyndon words and short superstrings. In: Proc. SODA’13. pp. 958–972 (2013)
23. Neuburger, S., Sokol, D.: Succinct 2D dictionary matching. *Algorithmica* pp. 1–23 (2012), 10.1007/s00453-012-9615-9
24. Provençal, X.: Minimal non-convex words. *Theor. Comput. Sci.* 412(27), 3002–3009 (2011)
25. Yamamoto, T., Bannai, H., Inenaga, S., Takeda, M.: Faster subsequence and don’t-care pattern matching on compressed texts. In: Proc. CPM 2011. LNCS, vol. 6661, pp. 309–322 (2011)
26. Ziv, J., Lempel, A.: Compression of individual sequences via variable-length coding. *IEEE Transactions on Information Theory* 24(5), 530–536 (1978)